Informe Proyecto Final

(Grupo: Martin Kondra, Agustin Fernandez, Emiliano Alvarez)

En el presente escrito intentaremos hacer un recorrido por lo que fue nuestro intento por resolver el problema de poder obtener un modelo de BERT que ante un texto y una pregunta, pueda predecir dónde está la respuesta esperada. Para lograr esto, utilizaremos los diferentes recursos adquiridos a lo largo del curso "Aprendizaje profundo" dictado por Luciano Robino y Pablo Marinozi, y coordinado por Matías Battocchia, integrantes del grupo "Datitos".

En primer lugar, nos parece necesario hacer una breve descripción de lo que es el punto de partida, a partir del cual, construiremos nuestro desarrollo, así como aquello que nos servirá de referencia para ver si lo implementado por nosotros implica una mejora sustancial en la tarea que estamos buscando resolver. Debemos destacar que gran parte del trabajo aquí realizado está basado en el documento: "CS 224N Default Final Project: Building a QA system (Robust QA track)" que es una guía para que estudiantes de la universidad de Stanford realicen su trabajo final en un curso con la misma temática que el que estamos finalizando nosotros. Ahora bien, dejando de lado estas formalidades, pasaremos a describir más en detalle el material con el que trabajaremos para luego explicar nuestros aportes.

Contaremos de antemano con un código, brindado previamente por los organizadores del curso, para preprocesar los datos y calcular las métricas de evaluación. A su vez, tendremos a nuestra disposición otro código que nos servirá de baseline(una variación de BERT llamada DistilBERT). Estos códigos, así como nuestros desarrollos posteriores, se entrenarán y validarán con una serie de datasets de diferentes características. Por un lado, tendremos tres datasets de conocimiento general que serán SQuAD, NewsQA, Natural Questions, con alrededor de 50000 ejemplos cada uno, y por el otro, tendremos otros tres, DuoRC, RACE, RelationExtraction, tres datasets de conocimiento específico con un tamaño mucho menor que los anteriores. La etapa de testeo únicamente se realizará sobre estos tres últimos. Ya teniendo en claro estas cuestiones, podremos avanzar en lo que fue desarrollo.

En primera instancia, es necesario que hagamos mención sobre los diferentes problemas que afrontamos a la hora de poner a entrenar los modelos utilizando el enviroment y todas las funciones y códigos provistos. Si bien contábamos con un servidor con GPU incorporado, la cantidad de ejemplos excedía nuestra capacidad de cómputo. Luego de varios intentos fallidos logramos entrenar nuestro Baseline con los tres datasets de conocimiento general. Para esto debimos seleccionar un número relativamente bajo Bath Size = 4.
Hagamos una pequeña mención a qué representan las métricas aquí arrojadas. Teniendo en cuenta que nuestro modelo no es un generador de respuestas en sentido estricto, sino que a partir de una pregunta intenta dar una respuesta a partir de lo que aparece en un texto preexistente, podremos entender tanto la métrica EM como la F1 a partir de un ejemplo. Si a nuestro modelo se le hiciera la siguiente pregunta, ¿Quién es el presidente de la Argentina?, y tuviese el siguiente fragmento de texto a partir del cuál encontrar la respuesta:

"El actual presidente es Alberto Fernández, de la alianza Frente de Todos, que tomó posesión el 10 de diciembre de 2019. "

Ahora bien, podremos entender las métricas antes mencionadas de la siguiente manera. EM (Exact Match) es una medida binaria (true/false), y su resultado depende de si el output de nuestro modelo se corresponde exactamente con la respuesta esperada. Es decir, si nuestro modelo respondiera "Alberto" en vez de responder "Alberto Fernández" (Que sería el target esperado a esa respuesta), el score de EM sería 0. Ahora bien, en cambio F1, es una métrica no tan estricta. Esta métrica es media armónica de precisión y recuperación (recall). En nuestro ejemplo, el modelo tendría una precisión del 100%(su respuesta es un subconjunto de la respuesta target) y un 50% de recuperación (solo incluía una de las dos palabras en la salida del target), por lo tanto, F1 tendrá un score de 2 × predicción × recuperación / (precisión + recuperación) = 2 ∗ 50 ∗ 100 / (100 + 50) = 66,67%.

Teniendo en cuenta esto, veremos que con el baseline y los 3 datasets de dominio general, las mejores métricas obtenidas sobre los dataset de Validación fueron:

F1= 70.70, EM = 54.95

Nuestra función de pérdida es la suma de la pérdida de probabilidad logarítmica negativa (entropía cruzada) para las ubicaciones inicial y final de nuestra predicción.) Para ser un baseline, parece tener un rendimiento aceptable. Nuestros resultados sobre los dataset de Testeo (se utilizaron los dataset correspondientes a la Validación sobre conocimiento específico - carpeta oodomain_val) fueron los siguientes:

F1:47.59, EM:31.94

A partir de aquí intentaremos explayarnos acerca del trabajo que realizamos intentando mejorar la performance del modelo.

Decidimos implementar "Mixture-Of-Experts (MoE)" para lograr nuestro cometido. Básicamente, y muy a grandes rasgos, podemos decir que esta idea consiste en entrenar una X cantidad de modelos “expertos” de BERT para dar respuesta a las preguntas planteadas, sopesando las respuestas de cada experto para construir una respuesta final a partir de la salida de una "Gatting Network".

Tomando como base el paper “R. Jacobs, Michael I. Jordan, S. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. Neural Computation, 3:79–87, 1991.” definimos los siguientes criterios de diseño:

la Gating Network sería un “encoder” BERT modificado de manera de que su salida (“vector de contexto”) se transforme en la selección de maestros
Tanto los maestros como la Gating Network se entrenarían simultáneamente y tendrían como entrada todos los ejemplos disponibles

Esquemáticamente sería:

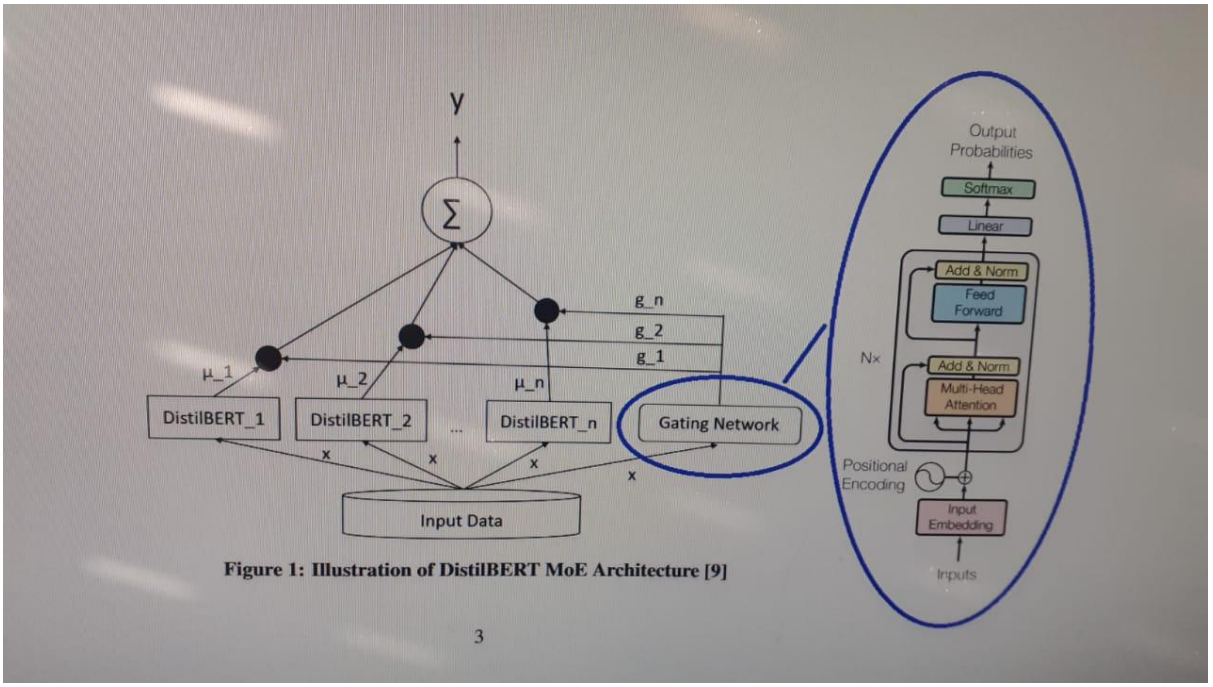


Figure 1: Illustration of DistilBERT MoE Architecture [9]

Gating Network:
Antes que nada, debemos comprender qué es esta "Gatting Network" a la que hemos mencionado en el párrafo anterior. Dado que el objetivo era que a partir de un texto de entrada seleccionáramos (o ponderáramos) una cantidad finita de Maestros, se nos ocurrió utilizar la parte del Encoder de los modelos Bert (Encoder-Decoder) sobre los que habíamos trabajado en el TP8, incorporando una capa densa extra para adecuar las dimensiones y un Softmax.

A esta Gating Network la llamamos Encoder_MoE.

En el Encoder_MOE intentaremos comprimir toda la oración fuente, X=(x1,...,xn) , en un solo vector de contexto, z. Este vector de contexto ha visto todos los tokens en todas las posiciones dentro de la secuencia de entrada. La representación del encoder después del último bloque se pasa a través de dos capas densas para llevar su dimensionalidad a [batch_size, cant_expertos].
Finalmente, mediante capa softmax asignamos los pesos que le corresponden a cada "experto" para cada ejemplo a procesar dentro de un batch.

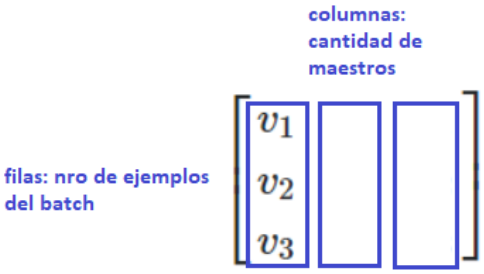
Dado que la Gating Network se entrenará en conjunto con los maestros, podemos decir que lo que está haciendo es que a partir de los textos de entrada (preguntas y contextos) genera una asignación de pesos a los maestros de forma tal que esto nos permita, posteriormente, determinar cuál es el modelo (experto) que llevará adelante la mejor performance frente a la tarea designada.

Respuesta única a partir de varios Maestros:
Ahora bien, cómo hacemos para que a partir de las salidas de todos nuestros expertos (en el caso puntual que nosotros diseñamos tiene 3 y la salida de la "Gatting Network" , construyamos una única respuesta para cada ejemplo del batch.

Este es un problema al que le encontramos una solución que consideramos bastante óptima tanto en términos matemáticos como computacionales.

Como ya mencionamos, inicialmente nos planteamos que la respuesta fuera una suma ponderada de las respuestas de todos los “expertos” y para esto desarrollamos un algoritmo que “construyera” nuestra nueva salida (suma ponderada de las salidas de cada maestro).

La salida de nuestra Gating Network nos entrega el peso asignado a cada Maestro para cada ejemplo del Batch. En la siguiente matriz la primer columna corresponde a pesos asociados al primer maestro. Por ejemplo, v2 es el peso asignado al primer Maestro para el segundo ejemplo del batch:



analizando la documentación de los modelos base utilizados (https://huggingface.co/transformers/model_doc/distilbert.html) obtuvimos los *Span-start scores* (before SoftMax) y *Span-end scores* (before SoftMax) ya que la idea era pesarlos de la siguiente manera (a continuación el ejemplo para el primer maestro):

pesos asignados por la gating network para el primer Maestro en cada ejemplo del batch

Span-scores (before SoftMax) del primer Maestro

$$v \odot F = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \odot \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \\ f_{3,1} & f_{3,2} & f_{3,3} \end{bmatrix} = \begin{bmatrix} v_1 f_{1,1} & v_1 f_{1,2} & v_1 f_{1,3} \\ v_2 f_{2,1} & v_2 f_{2,2} & v_2 f_{2,3} \\ v_3 f_{3,1} & v_3 f_{3,2} & v_3 f_{3,3} \end{bmatrix}$$

y lo anterior puede escribirse de la siguiente forma aplicando producto de Hadamard y vector de 1s.

$$v \odot F = v\mathbf{1}^T \circ F$$

y de manera de hacerlo fácilmente ejecutable y óptimo de aplicar durante el entrenamiento, puede implementarse de la siguiente manera:

$$v \odot F = \mathrm{Diag}(v)\, F$$

Luego sólo nos resta sumar elemento a elemento las matrices obtenidas para cada maestro y reconstruir las predicciones (start_logits y end_logits) realizadas por los modelos Distilbert.

Dado que ahora las salidas de cada modelo son pesadas según el gating network, debemos recalcular la pérdida (“loss”) del modelo MoE. Y eso lo hacemos replicando el código fuente de los modelos base utilizados. Código *DistilBertForQuestionAnswering* disponible en: https://huggingface.co/transformers/_modules/transformers/models/distilbert/modeling_distilbert.html#DistilBertForQuestionAnswering

Leyendo en detalle el paper citado (“Adaptive mixtures of local experts”) recomendaban utilizar un único maestro para obtener la salida, por lo tanto modificamos nuestro desarrollo para que se utilizara un sólo maestro en cada ejemplo de cada batch en lugar de hacer un pesaje de cada maestro para construir la salida unificada. Reemplazamos el Softmax() por un Max() con lo cual los pesos serán igual a 1 (uno) para el maestro con mejor puntuación y 0 (cero) para el resto.

La idea detrás de todo es que durante el entrenamiento los Maestros se especialicen en ciertas características/estructuras de los textos y sean mejores para obtener las respuestas según sea el caso, y que el Gating Network pueda decidir, según dicho texto de entrada, cuál es el Maestro que mejor performance obtendrá en su respuesta.

Si bien todo esto se oye muy lindo, la realidad es que a la hora de implementarlo tuvimos varias complicaciones, tanto a nivel funcionamiento como a nivel capacidad computacional para desarrollar un entrenamiento en condiciones óptimas. Lo segundo es quizás lo menos relevante a los fines últimos de nuestro trabajo pero en última instancia termina influyendo en las decisiones acerca de cómo estructurar el código. En este aspecto nos pasó que por más que contáramos con un server con GPU para realizar el entrenamiento, la capacidad de la GPU se veía desbordada cuando el del Batch Size superaba los 4. A base de prueba y error, dimos con esta solución de ir reduciendo el número del Batch Size hasta encontrar uno que funcionase. A su vez, esto también limitó el número de expertos que incluímos en nuestro desarrollo (sólo pusimos 3) ya que implica entrenarlos completamente a cada uno , así como también la cantidad de Data Frames que, por ejemplo, pudimos usar para correr el Baseline. De alguna forma esto fue un limitante para poder probar nuevas variantes que mejoraran realmente la labor del Baseline.

Ahora bien, en lo que respecta al código en sí y los problemas que surgieron, el más relevante fue que tal y como explicaba que podía pasar el paper que tomamos de referencia (R. Jacobs, Michael I. Jordan, S. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. Neural Computation, 3:79–87, 1991.), nuestro desarrollo no optaba naturalmente por entrenar de forma pareja a los expertos tal y como esperábamos que lo hiciera, sino que la Gating Network terminaba eligiendo siempre el mismo experto para todos los ejemplos. Ante este problema, el paper plantea que una solución posible es agregar una penalización que estimule a la Gating Network a elegir un número parejo de veces a cada experto durante el entrenamiento.

Nos costaba imaginarnos cómo implementar esa Función de Error Auxiliar (penalización) pero indagando llegamos a una función que se obtuvo de una implementación realizada en tensorflow basada en el paper "Sparsely-Gated Mixture-of-Experts Layers" (See "Outrageously Large Neural Networks" https://arxiv.org/abs/1701.06538:). Su función es adicionarle un valor auxiliar a la función de pérdida que penalice durante el entrenamiento si

la Gating Network selecciona siempre al mismo maestro. De esta manera se tienden a entrenar todos los maestros con la misma cantidad de ejemplos.

Finalmente, nuestro modelo entrenado utilizando la Mezcla de Maestros obtuvo los siguientes scores:
Validación/Test: **no logramos terminar una corrida de entrenamiento como para entregar los Scores aproximados.**

Análisis Final:

Lamentablemente no logramos correr nuestro modelo MoE totalmente por problemas de recursos computacionales, pero seguiremos intentando solucionarlo técnicamente porque creemos que el modelo que generamos tiene potencial.

Con las corridas parciales que logramos entendemos que el modelo no funciona correctamente aún porque nos falta encontrar la manera de que el optimizador actualice los parámetros de cada maestro de forma proporcional a la contribución que tuvo cada maestro en el cálculo final de la función de pérdida en cada batch.

Queremos probar:

estudiar cómo dirigir al optimizador (según algún tipo de criterio) para que realice las actualizaciones de los parámetros de cada maestro.

diferentes Learning Rates para los Maestros y la Gating Network.

analizar algún tipo de normalización de la función de error auxiliar (se suma a la función de pérdida antes de hacer el loss.backward() y luego el optim.step()) ya que debemos estudiar el impacto de la misma en el aprendizaje del modelo.

Gating Networks con diferentes complejidad (recordemos que en el Encoder que utilizamos pueden configurarse diferentes cantidades de capas de bloques de encoder, así como diferentes cantidades de cabezas de atención). Cambiando estos parámetros nuestros modelos, en teoría, se especializarían en diferentes características de los textos.

diferentes cantidades de Maestros. Si efectivamente lográramos que cada maestro se especializara en diferentes estructuras o tipos de textos, es muy interesante encontrar la cantidad óptima de maestros necesarias según el problema planteado.